ERDC MSRC PET Technical Report No. 01-27

A Prototype File Protocol for Application Data Sets Based on HDF

by

Kent E. Eshenberg
Mike Folk

24 May 2001

Work funded by the Department of Defense
High Performance Computing Modernization Program
U.S. Army Engineer Research and Development Center
Major Shared Resource Center through

Programming Environment and Training

Supported by Contract Number: DAHC94-96-C0002
Computer Sciences Corporation

Views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of Defense position, policy, or decision unless so designated by other official documentation.

A Prototype File Protocol for Application Data Sets Based on HDF

Kent E. Eschenberg, Ph.D.[1], and Mike Folk, Ph.D.[2]

Abstract The Hierarchical Data Format (HDF), Version 5, is the standard binary file format used by the Accelerated Strategic Computing Initiative (ASCI) as well as a great many other projects. A prototype protocol, UHDF, is under development that adds a layer upon HDF to provide support for several higher level concepts including time-varying data, unstructured cells, metadata about each component, and notes created during evaluation. One of the benefits of the new protocol will be that postprocessing tools, such as visualization packages, can more completely "understand" files without knowledge of the source of the file. In addition, by storing global parameters (such as the minimum and maximum of a parameter) in the metadata as the file is written, postprocessors will not need to read the entire data set to recalculate these parameters. Geometric scaling or color encoding could thus be based upon the entire data set even when working with only a subset.

1 Introduction

Upon the completion of a simulation, or the recording of data from an experiment, a series of tasks, called *postprocessing*, are begun to verify, analyze and, hopefully, understand the scientific events under study. In some cases, software tools for these tasks that are specific to the events under study are required. In other cases, tools can be shared. The extent of this sharing mainly depends upon whether the tool can intelligently access the file containing the data. The tool would need to be able to find out, for example, whether the data are structured or unstructured; whether the data set uses 1-D, 2-D, or 3-D coordinates; whether there are parameters at the nodes (vertices), cells, or both; and so on. A tool could be shared between a great many researchers in a field of study, or perhaps even across fields, if there were a file protocol that could span the needs for data storage across these areas.

Large projects may produce many files of results for different simulation or experimental conditions. Each file of results may be used to create additional files of derived parameters, subsampled arrays, and so on. The challenge of managing a large collection of files would be more easily met if there were a feature in the file protocol that could record information about the data, in addition to the data itself. For example, key features of the simulation and experimental conditions that were used to create the file would be recorded as well as the operations that have been performed on the data during post-processing.

Large projects may also involve several researchers working together to evaluate the results. While post-processing tools for analysis and visualization may help a researcher find areas of note, collecting these notes and ensuring that they are stored with the data file can be challenging. In some cases, forming the note itself can be tedious if the researcher needs to indicate a specific parameter, region of space, or span of time. Managing this information would be easier if the file protocol could record the note in the file containing the data.

A file protocol, UHDF, is proposed as a means to address these and other needs. The attached proposal gives a technical description of the protocol while this part of the paper focuses on underlying design

[1] U.S. Army Engineer Research and Development Center Major Shared Resource Center, Vicksburg, MS kee@erdc.hpc.mil

[2] National Center for Supercomputer Applications, University of Illinois at Urbana-Champaign mfolk@ncsa.uiuc.edu

issues, applications, and on plans to solicit the involvement of the scientific community in the refinement and implementation of the library.

2 Selecting The Foundations

Many scientific data formats are available in which structured and unstructured data could be described. A new layer of functionality such as UHDF could have been added to several of these. HDF5 uniquely meets the requirements of UHDF in a number of important ways.

Generality of the Format HDF5 includes the basic building blocks for creating the kinds of data structures that UHDF needs. It includes a grouping structure that makes it possible to organize and differentiate the entities that make up UHDF. It includes a structure that is a general multidimensional array structure that can be used for storing all of the types of data aggregates that UHDF needs. Taken together, these two structures make it possible to mix and match the entities needed to support UHDF in a flexible, straightforward way.

Flexible Access Methods The HDF5 application programming interface (API) and format are designed to enable very flexible access to data. As data sets become very large, UHDF applications being able to access portions of data sets efficiently becomes more and more important.

Flexible Storage The HDF5 format is really a collection of many different formats that can be suited to the needs of different applications. For instance, HDF5 data can be compressed and/or chunked to facilitate efficient storage and random access simultaneously.

Data Sharing HDF5 is a machine-independent format. The format itself contains full descriptions of its data, including the data types and how they are stored. The HDF5 library is able to convert the data stored in an HDF5 file to whatever format might be required on a particular platform. This capability is particularly important for UHDF because it will be common to create data on one platform, then analyze or visualize it on an entirely different platform.

Support for Many Platforms The HDF5 library runs on a very large number of computing systems, ranging from small workstations to the largest and fastest computers in the world.

Support for Large Data The HDF5 format and library are designed to enable applications to deal with very large data sets, which are becoming increasingly common in scientific and engineering applications. If UHDF is to handle grids with billions (or more) of cells, it must have a format in which to store the corresponding data.

High Performance I/O As applications migrate to clusters and other parallel computing environments, allowing different processors to access different parts of a data set in parallel becomes more important. The HDF5 library and format currently facilitate this by provide the capability for parallel I/O, and the NCSA HDF project is constantly working to improve this capability.

3 Applications Using UHDF

The UHDF library does not perform analysis or visualization; instead, it provides features that help the tools that are commonly used for these post processing tasks be efficient, reusable, and more useful. Tools written to use the UHDF library could provide several features.

Adaptation to Different Data Structures The UHDF file protocol collects information about the file's contents into a *datamap*. By examining this map, the tool could automatically adapt to the data structure,

array size, and parameters in the file (or announce why it cannot handle the particular case found in the file). A researcher embarking on a new project could save his results in the UHDF protocol and immediately benefit from a suite of UHDF-based tools.

Efficient Memory and Scale Management Using the datamap, a tool can determine the minimum and maximum sizes of each array thus enabling efficient memory management. A tool can also determine the minimum and maximum value, over all times, of a parameter so that a scaling for plots and colors can be calculated that are consistent over all time-steps. Without such features, tools generally must read all the data at all time-steps to determine these values.

More Informative File Browsers Many tools include a graphical user interface (GUI) that includes a dialog box for selecting the tool's input file. A tool designed to browse through UHDF files could enhance the dialog box to display, for each file, additional information. The information could be retrieved quickly using the datamap for each file and could include the file's source, the parameters listed by name, the number of time-steps, or any other information that the tool designer feels would be helpful to that application's users.

Coordinated Analysis and Visualization The UHDF protocol supports the concept of *data streams* and the inclusion of multiple data streams in a single file (please see the proposal for details). Briefly, the information in each data stream is that which is typically stored in a separate file. Each stream has its own structure, coordinates, and parameters. A tool for analysis or visualization could thus be given a single file name but be able to operate across all of the results of a simulation or experiment.

4 Design Issues

Many aspects of the design and implementation of the UHDF file protocol and library remain unresolved. While some are critical, the importance of other issues will depend upon the collective insights of the future users of the protocol.

Application Programmer Interface (API) Currently, the authors feel that the UHDF library itself should be implemented in C++. A goal of this project is that an API be provided for applications written in Fortran, C, and C++. For each, the manner in which a typical programmer in that language will wish to communicate with the UHDF library should guide the API design.

Tool and Library Interfaces Popular visualization tools such as AVS, EnSight, OpenDX, and The Visualization Toolkit (VTK) provide methods for extending their file readers. Readers for an earlier version of the UHDF library have been created for AVS and VTK with excellent results. Most of these packages provide data structures that parallel those of UHDF; however, an approach is needed for handling UHDF features that are not supported. The importance of each of these tools to the community of potential UHDF users should be determined and used to set development priorities.

Compression Many data fields include values that are null or otherwise unimportant (e.g., partially filled arrays). In other cases, lossless or lossy compression may be valid. Taking advantage of these aspects of the data could reduce the size of the file while increasing the challenge to make the UHDF library portable. This issue should be discussed to decide whether such compression should be handled by the application or by the UHDF library.

Subsets of Unstructured Data The HDF 5 protocol supports the selection of a subset of a structured array. A tool using such a feature could reduce the memory and time needed to focus on the area of interest to the researcher. Adding similar subsetting capabilities for unstructured fields would be useful

but could significantly increase the time and memory requirements. A variety of approaches are worth considering.

Extensions Currently, the UHDF library is envisioned as a self-contained package with new versions released as appropriate. Some may desire a means for adding features and algorithms without a need to upgrade the main UHDF library. Approaches, advantages, and disadvantages to such extensions should be carefully considered.

5 Summary

A new file protocol is under development that adds a layer upon HDF to provide support for several higher level concepts including time-varying data, unstructured cells, metadata about each component, and notes created during evaluation. The protocol UHDF supports efficient data storage, a high level of functionality, and fosters the reuse of tools from project to project.

Further development will be guided by the suggestions of the potential users of the file protocol. Hopefully, sufficient interest exists to justify the formation of a Web site for the dissemination of information about this effort and for the formation of a group to help resolve design issues and support the implementation of the library, API, and tool interfaces.

Acknowledgments

The authors gratefully acknowledge the contributions of the DoD High Performance Computing Modernization Program and the U.S. Army Engineer Research Development Center Major Shared Resource Center's Programming Environments and Training (PET) initiative.

A Proposal for Describing and Storing Structured and Unstructured Grids in HDF5

Kent Eschenberg[3] and Mike Folk[4]

June 20, 2001

6 Introduction

The Unstructured HDF (UHDF) library adds features to HDF5 for the support of data structures and processing that are often used in scientific research. A major motivation for this addition is the challenges that arise during the analysis and visualization (AV) that occur after a simulation (or experiment) has been completed. These processes may include the following:

- verification that the data and algorithm are valid
- reduction of the data to more easily focus on the key sections
- application of formulas to generate new parameters that are easier to evaluate
- interactive exploration using visualization

The UHDF library does not perform analysis or visualization; instead, it provides features that help the tools that are commonly used for these processes be more efficient and reusable. These features include the following:

(A) self-describing data fields
(B) quick access to data descriptions
(C) unstructured, time-varying data forms
(D) multiple data streams
(E) embedded file descriptions

Each of these features is motivated by a need that arises from the AV part of a project.

(A) The AV processes may vary from simulation to simulation and from the early to the latter stages of a project. This calls for a data format that approaches the ideal of being "self describing" so that the results from one process can be correctly read by the next, no matter the order of processes. The HDF5 library provides descriptions that include the dimensions and type of each data object. UHDF adds descriptions that link the data objects into a complete entity by identifying each as vertices, connections, node data, cell data, and so on.

(B) Analysis and visualization software can be major developments in their own right and so there is good reason to reuse the software throughout a project and from project to project. However, it is quite difficult for such software to efficiently accommodate a wide range of data set structures and features. UHDF assists by using the concept of a "datamap" that lists most features of a data stream and a data object. A visualization program, for example, could use the datamap to allocate memory and colors for the data using the datamap before reading the data itself.

(C) Simulations may produce "structured" data arranged in arrays of 1, 2 or 3 dimensions. HDF5 provides full support for storing such arrays using items of any HDF5 type. Simulations of physical processes may also need to use "unstructured" data consisting of a list of cells. UHDF adds a protocol for storing unstructured data with cells as simple as a point or as complex as a hexahedron. UHDF also adds support for structured and unstructured data objects that are dynamic (that vary in time) while continuing support for static objects.

(D) As the power and availability of computer systems increase, a greater number of simulations address a larger part of a physical process in order to better understand the interrelationship between the parts.

[3] U.S. Army Engineer Research Development Center Major Shared Research Center, Vicksburg, MS kee@erdc.hpc.mil

[4] National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign mfolk@ncsa.uiuc.edu

"Multiphysics" simulations are one method used to perform more comprehensive studies. While the data from each part could be stored in a separate file, management of the many files from a project is easier when the data from all the parts are stored together. UHDF supports such a method by introducing the concept of "data streams." Each stream may use a different data structure and time-steps.
(E) A project may conduct many simulations and produce many files. In addition, each "original" file may spawn a series of derived files as output from the analysis processes. In can be difficult to organize a large group of files produced over time by several members of the research team. UHDF provides a protocol for embedding "metadata" in the file to record the meaning of the file. Options include metadata about the source of the data, operations (such as reduction and formulas) that have been applied, and notes generated by software or a scientist. Applications that fully use these features produce files that are automatically identified.
The UHDF library, along with HDF5, could be used by a simulation to write its data streams directly to a UHDF file. In other cases, a separate program could be used to read a simulation's output and then rewrite the data using UHDF. Programs for analysis could use the library to read UHDF files as well as to write new UHDF files after, say, data reduction. Visualization programs could use the library to read UHDF files and to add notes that are made by a scientist during his interactive investigation.

7 Components of A UHDF Data Set[5]

There are three categories of information a UHDF file: metadata, datamaps, and application data. These categories have the following general meanings:

- The **metadata** category contains information about the source of the other data, about processing that has been applied to the data, about the current file, and other comments.
- The **datamap** category contains parameter values and other information for interpreting the other data, such as whether it is time-varying, what structures are involved, and how to interpret the data. The datamap is where other objects in the file are defined.
- The **application data** category contains (possibly large) arrays of numbers, including both **static** and **time-varying** data. The interpretation of this data is provided by the datamap. Data in the static group does not change through all time periods represented. Data in the time-varying or dynamic groups can change with each time-step.

7.1 Metadata

The metadata group contains background information about the data set. Metadata objects can contain values supplied by the application as well as values automatically calculated and supplied by an application, such as the UHDF library. All are stored as name-value pairs. UHDF, for instance, supports the following types of metadata.

- **Source Metadata**: information about the source of the data, such as the date, user, and name of a simulation or experiment
- **Filter Metadata**: information about processing that has already been applied to data, including date, user, name of processor, and information such as the original size of the data and span of coordinates before a subsetting filter was performed.
- **Current Metadata**: information about the current file, such as date, user, tool (and version) used to create the file, path to the original version of the file, and number of notes. It also includes information for reading the datamap such as the number of static and dynamic objects and the number of time-steps.
- **Notes Metadata**: Other comments attached to data by an application. Notes can be added to an existing file during evaluation and refers to one or more times, streams, or data objects. Each note can include text and "mark" objects. A mark object identifies spatial or temporal areas of

[5] A UHDF "data set" refers to the collection of objects in an HDF5 file that describe the data for an application. It is different from an HDF5 data set, which is a single HDF5 object.

interest; it could include a list of unstructured cells (points, edges, hexahedrons, etc.), a specific time or span of time, or variable or span of variables.

7.2 Datamaps

The datamap is designed so that an application, using the UHDF library to read a file, can discover a great deal about the file by first examining the datamap. This is particularly important when the application data arrays are extremely large. Information in the datamap could be used, for example, to allocate arrays, set color scaling, and make other preparations efficiently, all without having yet actually read the data. Because the datamap is a relatively small amount of information, the entire datamap can be read and converted to its in-memory form as soon as the UHDF library opens a file.

The application writing the file provides some of the information in the datamap, while the UHDF library automatically generates other information as the data are written. In general, an application writing a file using the UHDF library would not request information from the datamap, while an application reading from a file using the library would begin by requesting the entire datamap.

HDF5 inquiries can provide the dimensions of a particular array stored in a file. The datamap can provide additional information about the minimum and maximum size of the array, and minimum and maximum values in the array, over all times. It also describes how the array is used to create entities of a higher order called data streams. For example, a datamap records that an array of 100 vectors has a vector length of 3 (the X, Y, and Z of each vector), while the underlying HDF5 array sees only an array of 300 floats.

7.2.1 Data Streams

The concept of a **data stream** comes from of the idea that every experiment or simulation breaks down into several "systems" or "parts" that are monitored or computed, and that each part does not vary in its general structure over the course of the experiment or simulation. Each of these parts or systems has, or produces, a stream of data to be recorded. A data stream in a given file is a subset of the full menu of possibilities. It corresponds to the data that describes a simulation or an experiment and its results.

Data streams are the basic building blocks for much of the data in UHDF. A data stream is a uniform collection of related entities. It can include coordinates, connections, or solution data. A stream might be structured or unstructured cells, but not both. It may use hexagonal cells, or prism cells, but not a mix. It may consist of none or many parameters at each cell, and at each node, but the number of parameters does not change over time. Its coordinates may be static or may vary in time. Each of its parameters (attached to cells or nodes) may be static or may vary over time.

Data streams are also a good match to the way in which the information will be visualized. Typically, a different visualization technique will be selected for different data streams.

7.2.2 Forms

A data stream can have one, and only one, of several possible **forms**. Figure 1 shows the set of allowable data stream forms.[6] A given data set may have one or more data streams, and each data stream can use a different form.

[6] UML notation is used in some of the figures to indicate the relationship between an object and its children in a diagram, particularly to distinguish between aggregation ("part of") and generalization" ("is a") relations. A diamond indicates aggregation: the children are different parts of the parent. A triangle indicates generalization: each child is an instance of the parent, and only one child can be selected.. In the case of aggregation, some (perhaps all) of the children taken together make up the parent. In the case of generalization, only one of the children may exist.

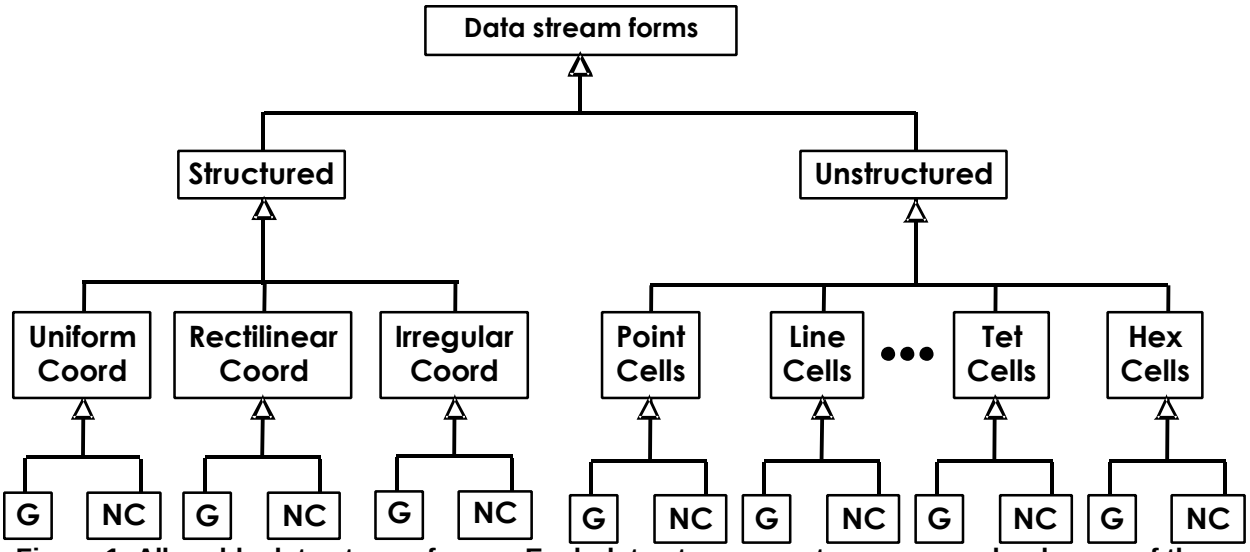

Figure 1. Allowable data stream forms. Each data stream must use one, and only one, of these forms. G stands for geometry data, and NC stands for node or cell data, respectively.

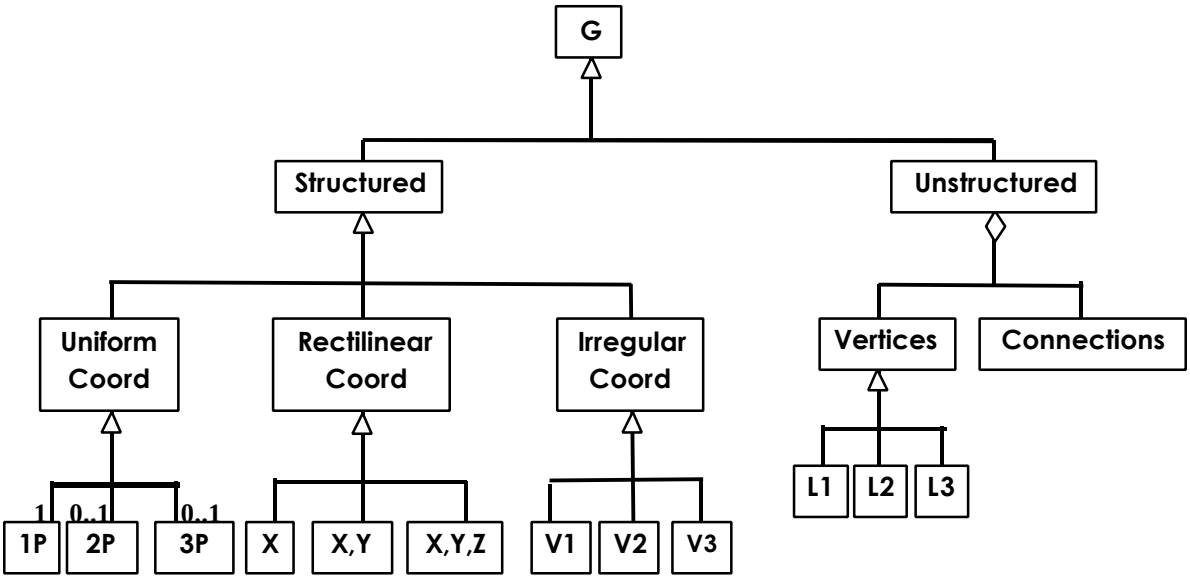

Figure 2. Geometry data options.

The geometry of a stream contains nodes and, if the stream is unstructured, the connections between the nodes that form cells. Streams also contain the parameter data that are associated with nodes or cells. In some systems the parameter data are referred to as variable data.

Geometry options are illustrated in Figure 2. Geometry data may describe structured or unstructured grids. All grids may be given for 1-D (one-dimensional), 2-D or 3-D space.

Structured grids are represented conceptually by 1-D, 2-D, or 3-D arrays. It is assumed that there is a node for every point in the array. For instance, if a 3-D grid has dimensions I, j, k, then there are I*j*k nodes. Three types of structured grids are shown: uniform, rectilinear, and irregular:

- In the case of **uniform grids**, the spacing between nodes is uniform. A uniform grid is described using up to three pairs of coordinate limits, signified by 1P, 2P, and 3P in Figure 2.

- For **rectilinear grids**, the spacing between coordinates can vary along any given axis. A rectilinear grid is described using 1, 2 or 3 arrays of coordinates, signified by the 1D arrays X, Y and Z in Figure 2.
- For **irregular grids**, the location of every node is explicitly specified. An irregular grid is described using a single array of coordinates with a vector length of one (X), two (XY), or three (XYZ) (denoted by V1, V2, and V3 in Figure 2). The dimensions of the array are the same as that of the grid itself.

The geometry data for **unstructured grids** describe the nodes and cells (polyhedra) that constitute the grid. The grid has two components: vertices (nodes) and connections (the indices of the nodes that form the cells). The index of the first node is zero.

The vertices are a one-dimensional array of floats with a vector length of one (X), two (XY), or three (XYZ) (denoted L1, L2, and L3 in Figure 2). The length of this array is equal to the number of nodes in the grid.

The connections are a one-dimensional array with a length equal to the number of cells in the grid. It has a vector length equal to the number of nodes needed for the data stream's cell type. For example, a stream consisting of 1,000 hexahedron cells would have an array of connections that was of length 1,000 with a vector length of 8. It would be stored as an HDF5 array of 8,000 integers.

7.2.3 Parameter Data

Figure 3 illustrates the options available for node data and cell data in each stream. There may any number of parameters for each node or cell, or none. Each parameter may have a vector length (number of values) from 1 to Nv and may be any one of the HDF5 data types. Any combination of data types, vector length, and number of parameters may be used.

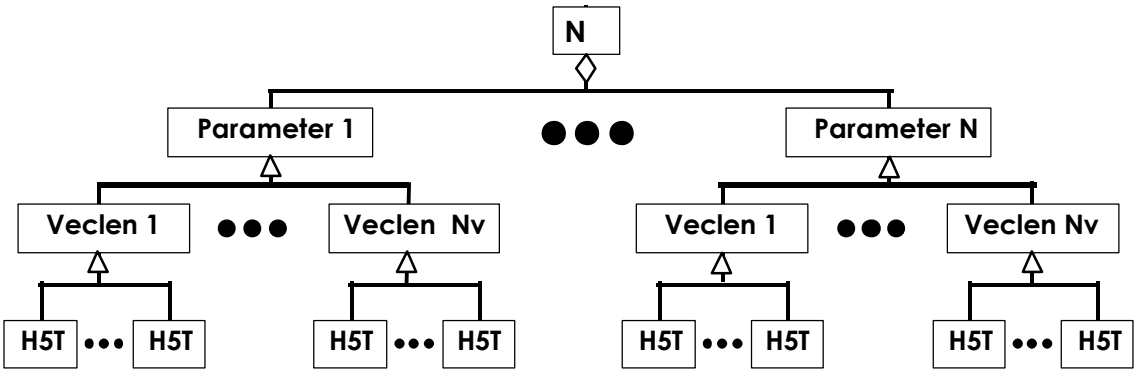

Figure 3. Node or cell data options.

7.2.4 Organization and Content

The datamap contains descriptions of application data objects. The application data objects themselves are not stored in the datamap but are stored with either the group of static, non-time-varying objects; or,

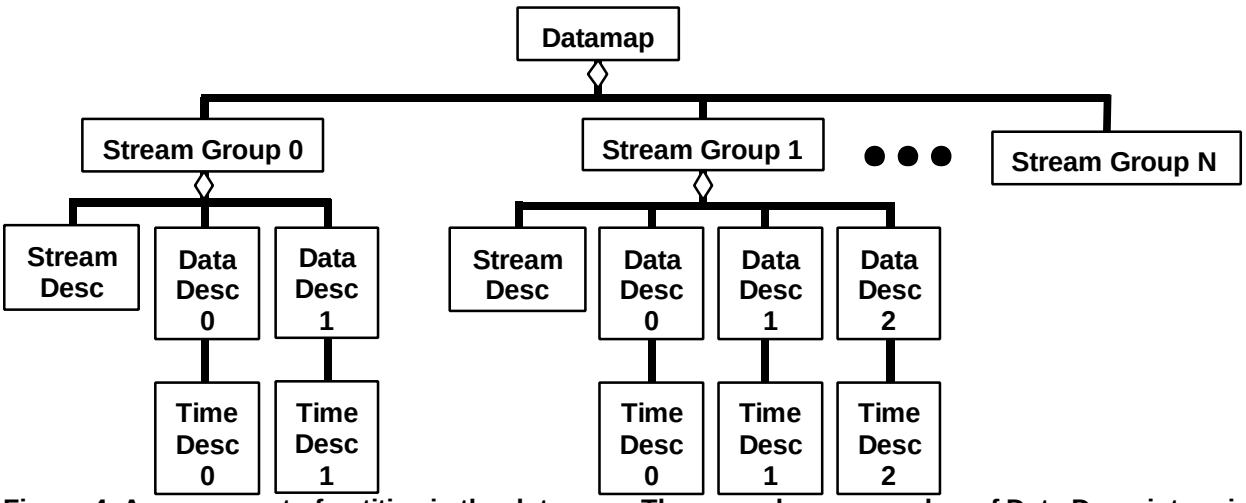

Figure 4. Arrangement of entities in the datamap. There can be any number of Data Descriptors in a Stream Group, and any number of Stream Groups.

with the group of objects for a particular time-step (see Figure 6). The datamap consists of three types of entities arranged in a hierarchy as shown in Figure 4.

A **stream group** has no information and serves only to indicate the entities that, together, constitute a data stream.

A **stream descriptor** contains information about the stream. It includes the following:

- identifier (an integer)
- name (long and short versions)
- number of data objects (an integer)
- form (structured or unstructured)
- cell type (point, …, tetrahedron; used only for unstructured forms)
- coordinate type (uniform, rectilinear, or irregular; used only for structured forms)
- array rank (1,2, or 3 for structured forms; always 1 for unstructured forms)
- space dimensions (1,2, or 3 meaning X, XY or XYZ)

A **data descriptor** contains information specific to one application data object in the data stream. This object could be, for instance, the vertices. One data descriptor applies to its application data object at all times. It includes the following:

- identifier (an integer)
- name (long and short versions)
- units (long and short versions)
- primary usage (grid, node or cell)
- secondary usage (vertices, connections, general data or material, interior or deleted flags)
- type (an H5T flag)
- vector length (number of values)
- min and max values for all times (an array whose length is the vector length)
- min and max dimensions for all times (an array whose length is the rank for this stream)
- number of time-steps at which the application data object is given (1 means it is static)

A **time descriptor** contains information about an application data object that may vary over time. The information includes the following:

- time-step (numbered starting at 0)
- application note 1 (set by application for any purpose)
- application note 2 (ditto)
- application data object dimensions

An object whose dimensions vary over time must meet additional requirements to be determined.

7.3 Data Stream Example

Figure 5 illustrates the concept of data streams. In this example, the effect of air and water temperature on pollution in a lake is simulated over a period of months and compared with samples taken at the actual lake. Even though the three streams are stored as one data set (i.e., one file), they make use of a variety of data structures, data attachments, and a mixture of static and time-varying (dynamic) data.

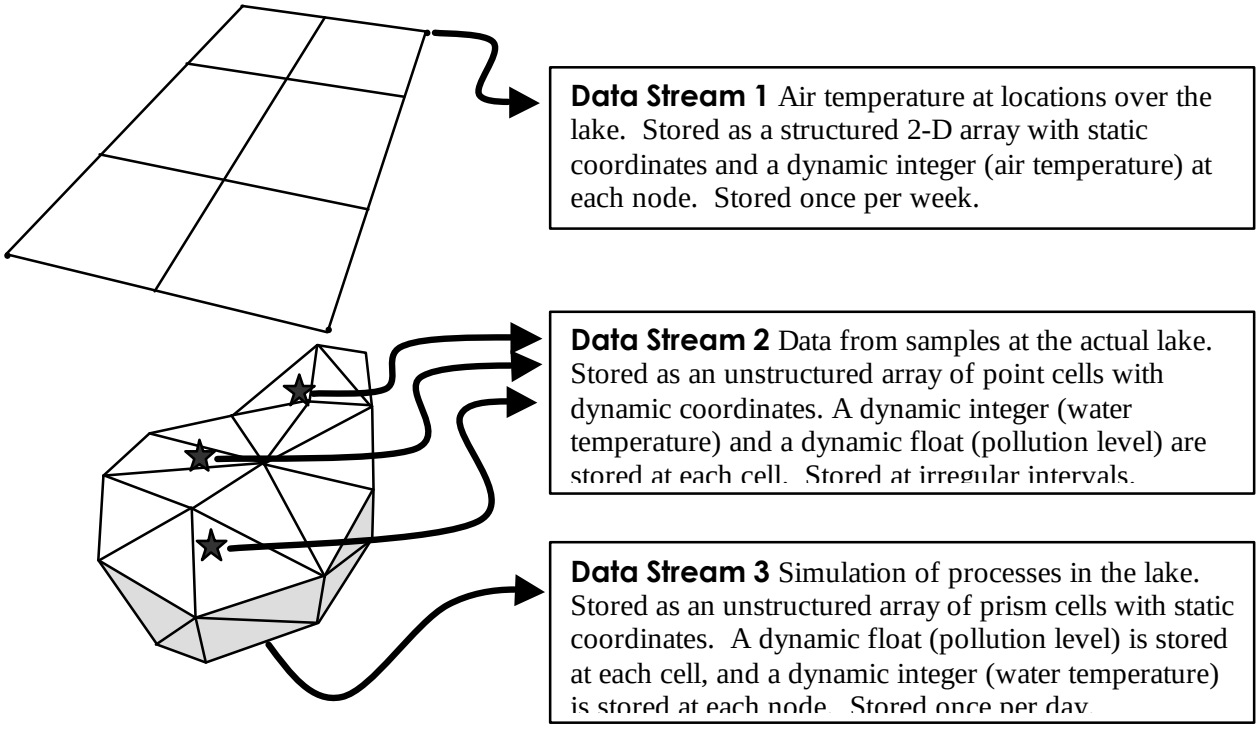

Figure 5. An example of data streams combining sampled data (air temperature and pollution level) with simulation results.

8 File Structure and API Layers

Any particular application that deals with structured and unstructured cell data has a certain view of the objects that it can deal with. It assumes that data are available in some accessible form and that certain operations can be performed on that data. The goal here is to provide sufficiently general structures and API in UHDF to support those of most unstructured cell data applications.

8.1 Data Structures

The data structure architecture that is being proposed consists of several layers, as illustrated in Table 1. Each layer provides information structures that can be used as building blocks to create structures at higher levels. The top layer represents the data structures dealt with by an application, such as VTK or EnSight. The second layer represents the data structures supported in the UHDF data model. The third

Application (VTK, EnSight, etc.)	Nodes, connectivity, variables, comments
UHDF	Metadata, datamap, data streams
HDF5	Data sets, attributes, groups
Data storage/network/memory	Bytes

Table 1. Data structure layers.

layer represents the structures used by HDF5, and the bottom layer represents the source or destination of data. The primary challenge is to define a set of UHDF structures that are sufficiently comprehensive that they can support the data needs of as many applications as possible.

In Section 2, it was mentioned that there are three categories of information in a UHDF file: metadata, datamaps, and application data. There is a natural mapping between these categories and the file structures of HDF5:

- The metadata and datamap information can be organized in HDF5 groups labeled "Metadata" and "Datamap," respectively.
- Static and time-varying application data can be organized in groups also. Static data can go in a group labeled "Static," and data for each time-step can go in a group labeled "T0," "T1," etc.
- The components of metadata group objects are pairs of strings.
- The components of static and time-varying application data are data streams.
- Data streams can be organized as groups of HDF5 data sets of homogeneous datatype.

This mapping is illustrated in Figure 6. "Metadata" and "Datamap" groups are placed under the top (root) group. Application data spans a number of groups, all stored under the root group. Static application data is stored in the group "Static", and time-varying data in a series of groups ("T1," "T2," etc.), one for each time-step.

8.1.1 Metadata

Each metadata object can contain values supplied by the application as well as values automatically calculated and supplied by the UHDF library. Most are stored as name-value pairs (i.e., as a pair of strings). The value string may use all characters including blanks. It may, for example, contain a short array of blank-separated numbers.

The syntax and semantics of the metadata are to be determined. Some metadata would be required, but much would be optional. For instance, filter metadata may be empty if no postprocessing has yet been applied.

8.1.2 Datamap

The datamap can be represented several ways even though the information stays the same. In a given application, there can be up to three representations:

- the version stored in the HDF file;
- the version stored in memory by the UHDF library; and
- the version delivered through the API for the selected language.

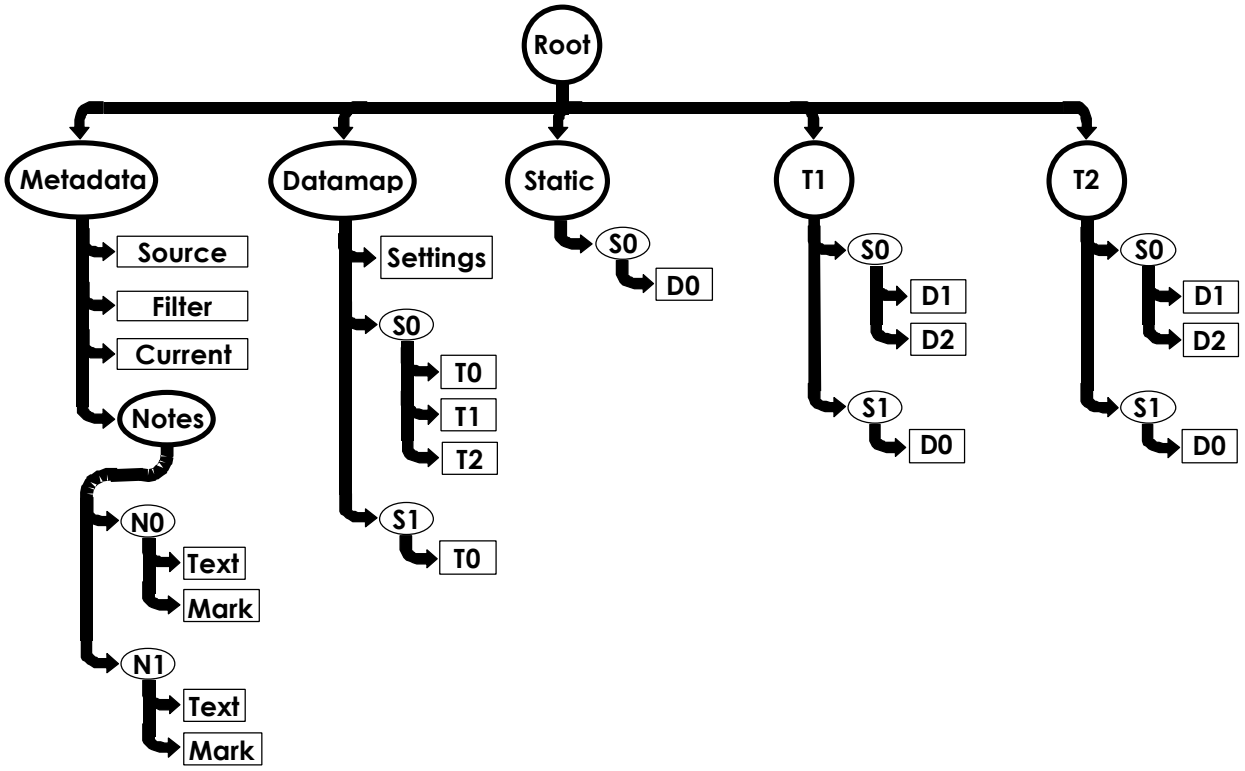

Figure 6. HDF entity organization used by the UHDF file format. Ovals and rectangles are group and data objects, respectively. Groups S0, S1, … are data streams. Objects D0, D1, … are application data objects such as vertices, cell connections, node data, and cell data. Groups N0, N1, … are note objects. Mark objects are a list of unstructured cells (points, lines, hexahedrons, etc.) designating an area of interest. Groups T1, T2, etc., contain data corresponding to distinct time-steps. There can be any number of times, streams, data objects, and notes.

Some language APIs merely provide pointers to the in-memory representation; otherwise, an application programmer need only know about the API representation.

All of the datamap except for the time descriptors is stored as an HDF5 object named **`/Datamap/Settings`**. It is a two-dimensional HDF array of strings where the fastest changing dimension is always two. That is, the array consists of pairs of strings called the name and the value. The name is composed of one or more parts separated by periods. The encoding of the value depends upon the value being stored but could be a string, a number, or an array of numbers. Only short arrays should be stored in this manner. A listing of the file version of a datamap is shown in Figures 8 and 9.

Each time descriptor is stored in a separate HDF5 object named **`/Datamap/Sx/Ty`** where **`x`** is the stream identifier and **`y`** is the data identifier. It is an array of integers of size **`NxM`** where **`N`** is the number of time-steps as given in the corresponding data descriptor. The value of **`M`** will vary from 4 to 6 with the elements containing the following information:

- **`[0]`** time-step (numbered starting at 0)
- **`[1]`** application note 1 (set by application for any purpose)
- **`[2]`** application note 2 (ditto)
- **`[3]`** application data object dimension 1
- **`[4]`** application data object dimension 2 (used only for data objects of rank 2 or 3)
- **`[5]`** application data object dimension 3 (used only for data objects of rank 3)

A particular application data object is not necessarily provided for all time-steps. For example, if an object of rank 2 is given for time-steps 3,4,5,10,11, and 12, the number of time-steps given in the object's data descriptor would be 6, and the array stored as the object's time descriptor would be of size 5 by 6. The leading part of a data descriptor name is the same as the HDF5 name used for the corresponding application data object. For example, information about application data object 4 in data stream 3 would be stored in the datamap using names that begin with **`S3.D4`**. The corresponding application data object would be stored, for example, at time-step 7 in an HDF5 object named **`/T7/S3/D4`**.

8.2 API

There are three levels of programmer interfaces for UHDF, as illustrated in Figure 7. Since the underlying file format for UHDF would be HDF5, all direct access to data is through the HDF5 API. The UHDF library provides a view of HDF5 objects consistent with the UHDF data types and structures described above. Given a UHDF API, one should then be able to write an interface to Ensight, AVS, VTK, etc., that can access the data in whatever way that is consistent with the requirements of the application.

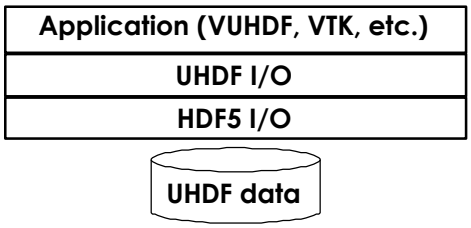

Figure 7. API Levels

8.2.1 UHDF API

Specifics of the UHDF API are to be determined. The purpose of this API would be to query, read, and write the objects described above. For example, an API would know all about cells and time-steps. It could find the minimum and maximum for entities and store them in a metadata section.

8.2.2 Using UHDF to Organize Application-Specific Data

Data structures for applications such as VTK and EnSight are to be specified using the UHDF API. Examples are to be determined.

```
S0.SD.id: 0
S0.SD.name.full: air simulation
S0.SD.name.trim: air
S0.SD.nobjects: 2
S0.SD.type: structured
S0.SD.coord: uniform
S0.SD.rank: 2
S0.SD.space: 3

S0.D0.id: 0
S0.D0.name.full: coordinates
S0.D0.name.trim: cord
S0.D0.units.full: meter
S0.D0.units.trim: m
S0.D0.usage.primary: grid
S0.D0.usage.secondary: vertices
S0.D0.type: float
S0.D0.veclen: 3
S0.D0.value.min: 0 0 10
S0.D0.value.max: 100.1 205 10
S0.D0.dim.min: 2 4
S0.D0.dim.max: 2 4
S0.D0.ntimes: 1

S0.D1.id: 1
S0.D1.name.full: air temp
S0.D1.name.trim: atmp
S0.D1.units.full: deg Celsius
S0.D1.units.trim: C
S0.D1.usage.primary: node
S0.D1.usage.secondary: data
S0.D1.type: int
S0.D1.veclen: 1
S0.D1.value.min: 13
S0.D1.value.max: 18
S0.D1.dim.min: 2 4
S0.D1.dim.max: 2 4
S0.D1.ntimes: 52

S1.SD.id: 1
S1.SD.name.full: field samples
S1.SD.name.trim: field
S1.SD.nobjects: 4
S1.SD.type: unstructured
S1.SD.cell: point
S1.SD.space: 2

S1.D0.id: 0
S1.D0.name.full: coordinates
S1.D0.name.trim: cord
S1.D0.units.full: meter
S1.D0.units.trim: m
S1.D0.usage.primary: grid
S1.D0.usage.secondary: vertices
S1.D0.type: float
S1.D0.veclen: 2
S1.D0.value.min: 17.3 22.9
S1.D0.value.max: 78.2 169.3
S1.D0.dim.min: 3
S1.D0.dim.max: 3
S1.D0.ntimes: 31

S1.D1.id: 1
S1.D1.name.full: connections
S1.D1.name.trim: conn
S1.D1.units.full: indices
S1.D1.units.trim: int
S1.D1.usage.primary: grid
S1.D1.usage.secondary: connect
S1.D1.type: int
S1.D1.veclen: 1
S1.D1.value.min: 0
S1.D1.value.max: 2
S1.D1.dim.min: 3
S1.D1.dim.max: 3
S1.D1.ntimes: 1

S1.D2.id: 2
S1.D2.name.full: water temp
S1.D2.name.trim: temp
S1.D2.units.full: deg Celsius
S1.D2.units.trim: C
S1.D2.usage.primary: node
S1.D2.usage.secondary: data
S1.D2.type: int
S1.D2.veclen: 1
S1.D2.value.min: 8
S1.D2.value.max: 12
S1.D2.dim.min: 3
S1.D2.dim.max: 3
S1.D2.ntimes: 31

S1.D3.id: 3
S1.D3.name.full: pollution
S1.D3.name.trim: pol
S1.D3.units.full: parts per million
S1.D3.units.trim: ppm
S1.D3.usage.primary: node
S1.D3.usage.secondary: data
S1.D3.type: float
S1.D3.veclen: 1
S1.D3.value.min: 2.3
S1.D3.value.max: 14.7
S1.D3.dim.min: 3
S1.D3.dim.max: 3
S1.D3.ntimes: 31

S2.SD.id: 2
S2.SD.name.full: water simulation
S2.SD.name.trim: sim
S2.SD.nobjects: 4
S2.SD.type: unstructured
S2.SD.cell: prism
S2.SD.space: 3
```

Figure 8. Datamap corresponding to example shown in Figure 5. Blank lines added for clarity.

```
S2.D0.id: 0
S2.D0.name.full: coordinates
S2.D0.name.trim: cord
S2.D0.units.full: meter
S2.D0.units.trim: m
S2.D0.usage.primary: grid
S2.D0.usage.secondary: vertices
S2.D0.type: float
S2.D0.veclen: 3
S2.D0.value.min: 2 4.1 1.5
S2.D0.value.max: 98.7 190 12
S2.D0.dim.min: 114
S2.D0.dim.max: 114
S2.D0.ntimes: 365

S2.D1.id: 1
S2.D1.name.full: connections
S2.D1.name.trim: conn
S2.D1.units.full: indices
S2.D1.units.trim: int
S2.D1.usage.primary: grid
S2.D1.usage.secondary: connect
S2.D1.type: int
S2.D1.veclen: 6
S2.D1.value.min: 0
S2.D1.value.max: 113
S2.D1.dim.min: 400
S2.D1.dim.max: 400
S2.D1.ntimes: 1
```

```
S2.D2.id: 2
S2.D2.name.full: pollution
S2.D2.name.trim: pol
S2.D2.units.full: parts per million
S2.D2.units.trim: ppm
S2.D2.usage.primary: cell
S2.D2.usage.secondary: data
S2.D2.type: float
S2.D2.veclen: 1
S2.D2.value.min: 2.2
S2.D2.value.max: 12.8
S2.D2.dim.min: 100
S2.D2.dim.max: 100
S2.D2.ntimes: 365

S2.D3.id.object: 3
S2.D3.name.full: water temp
S2.D3.name.trim: temp
S2.D3.units.full: deg Celsius
S2.D3.units.trim: C
S2.D3.usage.primary: node
S2.D3.usage.secondary: data
S2.D3.type: int
S2.D3.veclen: 1
S2.D3.value.min: 8
S2.D3.value.max: 13
S2.D3.dim.min: 114
S2.D3.dim.max:114
S2.D3.ntimes: 365
```

Figure 9. Datamap coresponding to example shown in Figure 5 (cont).

```
  0 0 0 2 4
  7 0 0 2 4
 14 0 0 2 4
    ...
357 0 0 2 4
/Datamap/S0/T1
```

```
  2 0 0 3
 24 0 0 3
 27 0 0 3
    ...
303 0 0 3
/Datamap/S1/T0
```

```
  0 0 0 114
  1 0 0 114
  2 0 0 114
    ...
364 0 0 114
/Datamap/S2/T0
```

```
  2 0 0 3
 24 0 0 3
 27 0 0 3
    ...
303 0 0 3
/Datamap/S1/T2
```

```
  0 0 0 100
  1 0 0 100
  2 0 0 100
    ...
364 0 0 100
/Datamap/S2/T2
```

```
  2 0 0 3
 24 0 0 3
 27 0 0 3
    ...
303 0 0 3
/Datamap/S1/T3
```

```
  0 0 0 114
  2 0 0 114
  2 0 0 114
    ...
364 0 0 114
/Datamap/S2/T3
```

.

Figure 10. Datamap time descriptors for example show in Figure 5.